Міністерство освіти і науки України

Чернігівський державний педагогічний університет імені Т.Г. Шевченка

Фізико–математичний факультет

Кафедра інформатики та обчислювальної техніки

Курсова робота

з математичного аналізу

Робота з таблицями засобами Lazarus

студента 25 групи

фіз-мат факультету

Майбороди Іллі Миколайовича

Чернігів, 2010

Засоби доступу до БД в середовищі Lazarus

Подібно до Delphi, Lazarus є середовищем RAD (від англ. Rapid application development - швидка розробка додатків), що дозволяє швидко створювати для користувача інтерфейс. На відміну від Delphi, Lazarus є кроссплатформенним середовищем. Підтримуються такі ОС, як GNU/Linux, Microsoft Windows, Mac OS X, FreeBSD, WinCE. Можливо перетворення Delphi проектів в проекти Lazarus.

Lazarus містить набір візуальних компонентів LCL (аналог VCL в Delphi). У версії 0.9.26 повністю підтримуються віджити Win32/Win64 та GTK, частково підтримуються віджити GTK2, QT, Carbon, Cocoa, WinCE і fpGUI. За кількістю компонентів Lazarus поки поступається Delphi. Тим не менш, є основний набір компонентів, а також компоненти для зв'язку з базами даних. IDE переведена на різні мови, в тому числі і на російську. Досить вибрати потрібну мову при установці Lazarus.

Lazarus підтримує такі бази даних:

* [dBase](http://traditio.ru/w/index.php?title=DBase&action=edit&redlink=1) та [FoxPro](http://traditio.ru/w/index.php?title=FoxPro&action=edit&redlink=1) за допомогою компонента TDbf
* [InterBase](http://traditio.ru/w/index.php?title=InterBase&action=edit&redlink=1) та [Firebird](http://traditio.ru/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F%3AAddData/%D0%98%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE/Firebird) за допомогою компонента TIBConnection
* [Oracle](http://traditio.ru/wiki/Oracle) за допомогою компонента TOracleConnection
* [PostgreSQL](http://traditio.ru/w/index.php?title=PostgeSQL&action=edit&redlink=1) за допомогою компонента TPQConnection
* [ODBC](http://traditio.ru/w/index.php?title=ODBC&action=edit&redlink=1) за допомогою компонента TODBCConnection.
* [MySQL](http://traditio.ru/w/index.php?title=MySQL&action=edit&redlink=1) за допомогою компонента TMySQL50Connection

Також підтримуються бази даних [MS SQL Server](http://traditio.ru/w/index.php?title=MS_SQL_Server&action=edit&redlink=1), для цього потрібно встановити бібліотеку [Zeoslib](http://zeoslib.sourceforge.net/index.php).

[DBase](http://traditio.ru/w/index.php?title=DBase&action=edit&redlink=1) та [FoxPro](http://traditio.ru/w/index.php?title=FoxPro&action=edit&redlink=1)

dBase - родина широко поширених систем управління базами даних, а також мова програмування, що використовується в них. Найперша СУБД цього сімейства називалася dBase II і була випущена в 1980 році компанією Ashton-Tate під CP/M, пізніше з'явилися версії для Apple II, Apple Macintosh, UNIX, VMS і IBM PC під DOS. Версія для PC разом з пришедшими їй на зміну dBase III і dBase IV були кілька років однією з найбільш розповсюдженіших програм. Довгий час dBase не портували під Microsoft Windows, в результаті чого у програми з’явилися сильні конкуренти - Paradox, Clipper, FoxPro та Microsoft Access. Оскільки формат даних dBase не був закритим, з середини 80-х безліч компаній стали виробляти свої діалекти мови і версії системи. У результаті з'явилося безліч схожих на dBase програм - FoxPro (сучасна Visual FoxPro), Arago, Force, dbFast, Clipper, Xbase + +, FlagShip, Recital, CodeBase, MultiBase, Harbour / xHarbour. Разом їх усіх іменують xBase.

[InterBase](http://traditio.ru/w/index.php?title=InterBase&action=edit&redlink=1) та [Firebird](http://traditio.ru/wiki/%D0%A1%D0%BB%D1%83%D0%B6%D0%B5%D0%B1%D0%BD%D0%B0%D1%8F%3AAddData/%D0%98%D0%B7%D0%B4%D0%B0%D1%82%D0%B5%D0%BB%D1%8C%D1%81%D1%82%D0%B2%D0%BE/Firebird)

В даний час останньою версією є InterBase 2009 (2009), у якій з'явилася підтримка Unicode та шифрування AES/DES. InterBase 7.5/2007 і Firebird 1.5/2.0 схожі, але вже далекі від повної сумісності - то є міграція між їхніми форматами баз даних легше, ніж між форматами зовсім «чужих» баз даних, але все-таки пов'язана з певними проблемами.

Основними перевагами останньої версії InterBase є низькі вимоги до системи, з одночасною масштабність на кілька процесорів, плюс розвинена система моніторингу, тимчасові таблиці, що вбудовується аутентифікація користувачів, журнал роботи. Традиційним гідністю вважається крос-платформеність – InterBase підтримує GNU/Linux, Microsoft Windows, Unix і Solaris.

[Oracle](http://traditio.ru/wiki/Oracle)

Перша у світі база даних, розроблена спеціально для роботи в мережах розподілених обчислень Grid, призначена для ефективного розгортання на базі різних типів обладнання, від невеликих серверів до потужних симетричних багатопроцесорних серверних систем, від окремих кластерів до корпоративних розподілених обчислювальних систем. СУБД надає можливість автоматичного налаштування і управління, яка робить її використання простим і економічно вигідним.

PostgreSQL

Опції є блоками коду, що виконуються на сервері, а не на клієнті бази даних. Хоча вони можуть писатися на чистому SQL, реалізація додаткової логіки, наприклад, умовних переходів і циклів, виходить за рамки власне SQL і вимагає використання деяких мовних розширень. Функції можуть писатися з використанням однієї з наступних мов:

* Вбудований процедурний мову PL/pgSQL, багато в чому аналогічний мови PL / SQL, що використовується в СУБД Oracle;
* Скриптові мови - PL/Lua, PL/LOLCODE, PL/Perl, pIPHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl і PL/Scheme;
* Класичні мови - C, C + +, Java (через модуль PL/Java);
* Статистична мова R (через модуль PL/R).

PostgreSQL допускає використання функцій, що повертають набір записів, який далі можна використовувати так само, як і результат виконання звичайного запиту. Функції можуть виконуватися як з правами їх творця, так і з правами поточного користувача. Іноді функції ототожнюються з збереженими процедурами, однак між цими поняттями є різниця.

[ODBC](http://traditio.ru/w/index.php?title=ODBC&action=edit&redlink=1)

[ODBC](http://traditio.ru/w/index.php?title=ODBC&action=edit&redlink=1) це програмний інтерфейс ([API](http://ru.wikipedia.org/wiki/API)) доступу до баз даних. Дозволяє однаково оперувати з різними джерелами даних, абстрагуючись від особливостей взаємодії в кожному конкретному випадку. При застосуванні ODBC потрібно пам'ятати, що дана технологія доступу до даних не розрахована на роботу з великою кількістю клієнтів. У тому випадку, якщо необхідно, щоб з базою даних одночасно працювало багато активних клієнтів, потрібно використовувати SQL API або спеціальний інтерфейс для взаємодії з конкретною базою даних.

Створення і робота з таблицями за допомогою [DBase](http://traditio.ru/w/index.php?title=DBase&action=edit&redlink=1)

Робота з базами даних типу [DBase](http://traditio.ru/w/index.php?title=DBase&action=edit&redlink=1) реалізується за допомогою компоненти TDbf. TDbf забезпечує доступ до таблиць баз даних dBase і FoxPro для Lazarus (та інших IDE), з огляду на читання, запис і створення dBase III+, dBase IV, Visual dBase VII і FoxPro таблиць. Все це робиться без використання додаткових бібліотек або движків бази даних. Просто розмістіть компонент TDbf на вашій формі, і Ви маєте миттєвий доступ в навколишньому середовищі бази даних на різних платформах.

TDbf є безкоштовним компонентом прямого доступу, для сумісних середовищ фірми Borland Delphi language. Це включає Delphi, C ++ Builder і Kylix. Він дозволяє створювати дуже компактні програми роботи з базою даних, яка не вимагає застосування програм інсталяції. Код роботи з базою включається до виконуваної код. Це забезпечує наступні можливості:

* Робота без BDE.
* Дозволяє використовувати рідні dBase типи (character, numeric, logical, date і memo). Див властивість TableLevel.
* Підтримує обидва типи Memo полів - text і binary, так можна використовувати поля без обмеження розміру.
* Формат файлів на 100% сумісний з dBase III + or dBase IV або dBase for Windows.
* Підтримка таблиць Clipper і Visual FoxPro.
* Модифікація існуючих таблиць включає видалення, додавання або модифікацію поточної таблиці без втрати даних.
* Багато користувальницький доступ сумісний по блокуванню з BDE. Тільки один користувач може блокувати запис для запису, але інші користувачі можуть читати цей запис.
* Підтримка індексів для швидкого сортування, пошуку і виділення діапазону на великих таблицях. Підтримка індексів для NDX і MDX індексних файлів.
* Вирази підтримуються як для індексів, так і для фільтрів.

TDbf працює і в Windows і в Linux використовуючи Lazarus. Оскільки в Lazarus немає програми на зразок "Database Desktop", ми повинні створити нову базу даних в коді.

Гарною думкою є створення для БД окремого каталогу. Це спростить резервування даних. Є два варіанти встановлення шляху. Ви можете встановити повний шлях використовуючи властивість FilePathFull, або встановити шлях щодо варіанти застосування за допомогою властивості FilePath. Наприклад, інсталяція "FilePath" під час виконання до "data/" використовуватиме субдиректорію data тільки в директорії здійснимих файлу. Інсталяція властивості "FilePathFull" до "/var/data/" буде розміщувати точно за вказаною шляху, ігноруючи розташування програми.

За замовчуванням, TDbf створює таблиці dBase IV. Хоча це і найбільш сумісно, деякі можливості, які Вам можуть знадобитися, не підтримуються. Для підтримки автоінкрементних полів ви повинні використовувати дещо більш нове. Типи таблиць можуть бути наступними:

* 3 dBase III+
* 4 dBase IV
* 7 Visual dBase VII
* 25 FoxPro

Створення полів для Вашої нової таблиці під час виконання у більшості випадках слідує старому стандарту Delphi. Як тільки ви встановили свої значення для властивостей FilePath, TableLevel, і TableName, маніпулюйте властивістю FieldDefs щоб визначити структуру. Наприклад:

MyDbf.FilePathFull:= ‘/location/to/my/data’;

MyDbf.TableLevel := 7;

MyDbf.TableName := ‘customers.dbf’;

With MtDbf/FieldDefs do

Begin

 Add(‘Id’,ftAutoInc,0,True);

 Add(‘Name’,ftString,80,True);

End;

Типи полів визначено як:

* ftUnknown
* ftString
* ftSmallInt
* ftInteger
* ftWord
* ftBoolean
* ftFloat
* ftCurrency (TableLevel 25)
* ftBCD (TableLevel 25)
* ftDate
* ftTime
* ftDateTime
* ftBytes (TableLevel 25)
* ftVarBytes
* ftAutoInc (TableLevel 7 or 25)
* ftBlob
* ftMemo
* ftGraphic
* ftFmtMemo
* ftParadoxOle
* ftDBaseOle
* ftTypedBinary
* ftCursor
* ftFixedChar
* ftWideString
* ftLargeInt
* ftADT
* ftArray
* ftReference
* ftDataSet
* ftOraBlob
* ftOraClob
* ftVariant
* ftInterface
* ftIDispatch
* ftGuid
* ftTimeStamp
* ftFMTBcd

Жирним виділені типи полів які підтримуються.

Як тільки ви визначили поля, які бажаєте використовувати у вашій новій таблиці, ви можете йти далі і створити таблицю:

MyDbf.CreateTable;

Якщо Ваша база даних складається з великої кількості записів, Вам може знадобитися визначити індекси щоб здійснювати більш швидкий пошук по таблиці. Щоб змінити структуру індексів таблиці, ми повинні мати ексклюзивний доступ до таблиці - який ми так чи інакше мали б, створюючи таблицю.

MyDbf.Exclusive := True;

MyDbf.Open;

Тепер, нам необхідно просто додати індекси:

MyDbf.AddIndex(‘custid’,’Id’,[ixPrimary,ixUnique]);

MyDbf.AddIndex(‘cutname’,’Name’,[ixCaseInsensitive]);

MyDbfClose;

Наступний приклад створює нову таблицю "customers" в коді. Звичайно, це необхідно виконати тільки один раз, і після цього всього лише ВІДКРИВАТИ таблицю, не створюючи її заново:

{$MODE OBJPC}

Program DataBaseTest;

Uses Dbf, db, Dbf\_Common;

Var

 MyDbf : Dbf;

Begin

 MtDbf := TDbf.Create(nil);

 Try

 MtDbf.FilePath := ‘data/’;

 MtDbf.TableLevel := 7;

 MtDbf.Exclusive := True;

 MtDbf.TableName := ‘customers.dbf’

 With MtDbf.FieldDefs do

 Begin

 Add(‘Id’,ftAutoInc,0,True);

 Add(‘Name’,ftString,80,True);

End;

MtDbf.CreateTable;

MtDbf.Open;

MyDbf.AddIndex(‘custid’,’Id’,[ixPrimary,ixUnique]);

MyDbf.AddIndex(‘custname’,’Name’,[ixCaseInsensitive]);

MyDbfClose;

Finally

 MyDbf.Free;

End;

End;

TDbf також підтримує зберігання вторинних індексів в окремих файлах. Це може допомогти в разі якщо буде дуже великий розмір бази даних. Файли вторинних індексів створюються ідентично звичайним, але з додаванням розширення файлу '. ndx':

MyDbf.AddIndex(‘custname.ndx’,’Name’,[ixCaseInsensitive]);

Кожного разу при відкритті TDbf, необхідно завантажити файл індексів:

MyDbf.OpenIndexFile(‘custname.ndx’);

Також індекси повинні бути описані включаючи розширення:

MyDbf.IndexName := ‘custname.ndx’;

Індексні файли можуть бути упаковані простим використанням:

MyDbf.CompactIndexFile(‘custname.ndx’);

Приклад, описаний раніше, показує, як створити нову таблицю бази даних програмно. Використання цієї таблиці більш ніж просте. Компоненти доступу до даних у Lazarus (як наприклад TDbEdit) вказують на компонент TDataSource використовуючи свої властивості "DataSource" і "DataField". Компонент TDataSource обробляє сполучення між движком бази даних і компонентами доступу до даних. Потім компонент TDataSource вказує на компонент TDbf використовуючи його властивість "DataSet". Зв'язок нагадує таке:

Переконайтеся, що Ви встановили властивості FilePath (або FilePathFulll), TableLevel, і TableName вашого компоненту TDbf перед викликом:

TDbf.Active := True;

Коли запис видаляється, насправді він фізично залишається в таблиці. Періодично Ви повинні "пакувати" таблицю щоб відновити втрачене місце. Це може бути зроблено установкою ексклюзивного режиму доступу:

MyDbf.Exclusive := True;

MyDbf.Open;

MyDbf.PackTable;

//давайте, також, переробимо індекси

MyDbf.Regenerate.Indexes;

MyDbf.Close;

MyDbf.Exclusive := False;

Справжня сила програмування баз даних починається коли Ви маєте декілька таблиць які посилаються один на одного. Поки TDbf не підтримує довідкову цілісності, але підтримує подання головна/підпорядкована між таблицями TDbf.

Якщо ви хотіли показати всі накладні для наданого клієнта, підпорядкована таблиця (invoices) може залишатися синхронізованою з головною таблицею (customers) автоматично.

На компоненті накладні TDbf встановлюють наступне:

InvDbf.IndexName := ‘idxcustid’;

InvDbf.MasterSource := dsCustomers;

InvDbf.MasterFields := ‘Id’;

Можливості TDbf на цьому не закінчуються. Для більш складних програм знадобляться багато інших властивостей TDbf. Нижче розглянемо всі методи та властивості компонента TDbf.

1. Методы

* 1. GetFieldData

function GetFieldData(Field:TField; Buffer:Pointer): Boolean;override;

Більшості додатків не потрібно викликати GetFieldData. Об'єкт TField викликає цей метод, через реалізацію в методі GetData. Параметр Field або FieldNo вказує поле, з якого мають бути вилучені дані. Параметр Field вказує на сам компонент, у той час як FieldNo вказує на номер поля. Параметр Buffer це буфер в пам'яті, який повинен бути достатнього розміру, щоб помістити значення поля, так як вона подана в базі (не відформатовано і не перетрансльовано). Параметр NativeFormat вказує, що компонент dataset витягує поле в рідному форматі C Builder для зазначеного типу поля. Коли NativeFormat дорівнює false, компонент dataset повинен конвертувати значення поля в належний тип. Це дозволяє обслуговувати дані з наборів різних типів (ADO, BDE або інших) витрат платника податку. GetFieldData повертає значення вказує, що дані були витягнуті успішно. GetFieldData повертає true, якщо буфер успішно заповнений даними з поля, і false, якщо дані не були витягнуті.

1.2 Resync

procedure Resync(Mode: TResyncMode); override;

TDbf підтримує заборону виклику resync.

Див. властивість DisableResyncOnPost.

1.3 CreateBlobStream

function CreateBlobStream(Field: TField; Mode: TBlobStreamMode): TStream; override; {virtual}

Викличте CreateBlobStream для створення потоку (stream) для читання з BLOB поля або для запису в нього (binary large object BLOB).Параметр Field повинен вказувати на компонент TBlobField з властивості Fields. Параметр Mode вказує, як використовується потік - для читання, для запису або для оновлення вмісту поля. Потоки Blob створюються в зазначеному режимі для конкретної запису. Додатки повинні створювати новий потік щоразу, коли змінюється запис, замість повторного використання існуючого потоку.

* 1. Translate

{$ifdef DELPHI\_4}

 function Translate(Src, Dest: PChar; ToOem: Boolean): Integer;

 override; {virtual}

{$else}

 procedure Translate(Src, Dest: PChar; ToOem: Boolean); override; {virtual}

{$endif}

Дані записуються в DBF файл в конкретній кодової сторінці, "OEM". Windows використовує кодові сторінки "ANSI" для показу даних. Ця функція перетрансльовує між цими кодовими сторінками. Вказівка true в ToOem транслює з Windows в DBF. Вказівка false в ToOem транслює з DBF в Windows.

1.5 ClearCalcFields

procedure ClearCalcFields(Buffer: PChar); override;

Внутрішній метод.

1.6 CompareBookmarks

function CompareBookmarks(Bookmark1, Bookmark2: TBookmark): Integer; override;

Викличте CompareBookmarks для визначення того, що обидві закладки (bookmarks) ідентичні чи ні. Параметри Bookmark1 і Bookmark2 це закладки для порівняння. Якщо закладки розрізняються, то CompareBookmarks повертає 1. Якщо закладки ідентичні або обидві закладку рівні NULL, то CompareBookmarks повертає 0.

1.7 ChekDbfFieldDefs

procedure CheckDbfFieldDefs(DbfFieldDefs: TDbfFieldDefs);

Робить перевірку на допустимість типів полів ваших визначеннь в TDbfFieldDef на відповідність поточному стану TableLevel. При використанні TableLevel менш ніж 7, не всі типи є допустимими.

* 1. AddIndex

{$ifdef DELPHI\_5}

procedure AddIndex(const AIndexName, Fields: String;

Options: TIndexOptions; const DescFields: String = '');

{$else}

procedure AddIndex(const AIndexName, Fields: String;

Options: TIndexOptions);

{$endif}

Параметр AIndexName це ім'я нового індексу. AIndexName повинен містити ім'я індексу, довжиною не більше 10 символів.

Параметр Fields це значення типу AnsiString value містить поле або вираз на основі якого базується новий індекс.

Параметр Options це безліч атрибутів для індексу. Параметр Options може містити одне, декілька або жодного значення TIndexOptions: ixPrimary, ixUnique, ixDescending, ixCaseInsensitive і ixExpression.

1. . Значення ixPrimary вказує на справді унікальний індекс. При спробі вставки двох однакових ключів буде порушено виняток.
2. . Значення ixUnique вказує на унікальний індекс. Повторні ключі ігноруються.
3. . Значення ixDescending вказує зворотний порядок сортування.
4. . Значення ixCaseInsensitive не використовується.
5. . Значення ixExpression не обов'язково вказувати, це визначається автоматично при розборі параметра Fields.
	1. RegenerateIndexes

procedure RegenerateIndexes;

Очищає всі підключені індекси і потім створює їх з нуля.

1.10 CancelRange

procedure CancelRange;

Викличте CancelRange для видалення встановленого діапазону в поточній таблиці. Видалення діапазону відновлює доступ до всіх записів набору даних.

1.11 SearchKey

function SearchKey(Key: Variant; SearchType: TSearchKeyType): Boolean;

function SearchKeyPChar(Key: PChar; SearchType: TSearchKeyType): Boolean;

Ця функція припускає, що Ви використовуєте на належний індекс, властивість IndexName. Параметр Key вказує значення для пошуку з активного індексу. Ви можете вказати ключ, як варіантний тип або передати буфер, використовуючи функцію SearchKeyPChar. У даному випадку, передача буфера повинна відповідати таким правилам, заснованим на індексі і типу параметра key:

* Рядковий індекс: вказівник на перший символ рядка, обмежений нулем.
* MDX, цифровий: покажчик на буфер, який містить BCD, у форматі dBase.
* NDX, цифровий: покажчик на тип double.

Параметр SearchType це одне з наступних значень:

* Значення stEqual пошук точної відповідності ключа. Повертає false, якщо ключ не знайдено.
* Значення stGreaterEqual пошук точної відповідності ключа або якщо не знайдено, то такий запис по порядку, яка більше за значенням, ніж ключ. Повертає false, якщо досягнуто кінець файлу.
* Значення stGreater пошук першого запису, значення якої більше значення ключа. Повертає false, якщо досягнуто кінець файлу.

Якщо повертається значення false то вказівник не рухається.

1.12 SetRange

procedure SetRange(LowRange: Variant; HighRange: Variant);

procedure SetRangePChar(LowRange: PChar; HighRange: PChar);

Ця функція припускає, що Ви використовуєте на належний індекс, властивість IndexName. Функція встановлює діапазон записів для поточного набору записів. Параметр LowRange вказує початкову границю, а параметр HighRange вказує кінцеву. Для форматування параметра дивіться функцію SetRangePChar і SearchKeyPChar.

1.13 UpdateIndexDefs

procedure UpdateIndexDefs; override;

Внутрішній метод, який викликає оновлення визначень полів (FieldDefs), оновлює обидва визначення для полів і для індексів, читаючи їх з dbase і індексних файлів.

1.14 GetIndexNames

procedure GetIndexNames(Strings: TStrings);

Параметр Strings вказує список, який при поверненні буде містити список всіх індексів. Значення з цього списку можуть бути використані для установки властивості IndexName.

1.15 TryExclusive

procedure TryExclusive;

Вимагає установки властивості Active в true. Викличте TryExclusive для отримання монопольного доступу до відкритого файлу, без виклику Close, встановлює властивість Exclusive в значення true і заново відкриває таблицю. Властивість Exclusive оновлюється для відображення нового стану. Перевірте властивість Exclusive для того, щоб переконатися, що встановлення режиму пройшла успішно.

1.16 EndExclusive

procedure EndExclusive;

Якщо Ви працювали в монопольному режимі, то виклик EndExclusive повертає вас у попередній стан.

1.17 LockTable

function LockTable(const Wait: Boolean): Boolean;

Викличте функцію LockTable для блокування всієї таблиці. Параметр Wait вказує, що компонент повинен чекати установки блокування або відмови, якщо блокування не вдається встановити.

Різниця між режимами LockTable і Exclusive полягає в тому, що в режимі Exclusive інші не можуть відкрити файл за винятком, коли властивість ReadOnly встановлено у true, а для LockTable вони можуть відкривати файл в режимі читання/запису. У режимі LockTable спроби зміни записів будуть відкинуті.

1.18 UnlockTable

procedure UnlockTable;

Коли таблиця була заблокована за допомогою LockTable, виклик UnlockTable призведе до розблокуванні табліци.

## 1.19 OpenIndexFile

procedure OpenIndexFile(IndexFile: string);

## Викличте процедуру OpenIndexFile для приєднання індексного файлу (IndexFile), вторинний, що не обслуговуються індексний файл, наприклад NDX файл до DBF файлу. Як тільки індексний файл приєднується, то він відразу стає обслуговується.

## 1.20 DeleteIndex

procedure DeleteIndex(const AIndexName: string);

Параметр AIndexName повертає назву індексу, який потрібно видалити.

* Якщо індекс міститься в MDX файлі, то він видаляється звідти.
* Якщо індекс міститься в NDX файлі, то він закривається, від'єднується і видаляється з диска.

1.21 CloseIndexFile

procedure CloseIndexFile(const AIndexName: string);

## Індексний файл, відкритий функцією OpenIndexFile або установкою властивості Indexes можна закрити за допомогою функції CloseIndexFile. Вказаний індекс більше не буде обслуговуватися.

## 1.22 RepageIndexFile

procedure RepageIndexFile(const AIndexFile: string);

Іноді при перегляді розміру індексного файлу можна помітить, що розмір його не зменшується після видалення індексу. Ця функція перебудовує вказаний індексний файл з метою зменшення розміру. Передайте, порожній рядок для перебудови MDX файлу таблиці. Ефект від застосування RepageIndexFile або пересозданія всіх індексів в індексному фото однаковий, але RepageIndexFile працює значно швидше. Процедура RepageIndexFile аналогічна операції "PackTable", але для зазначеного індексного файлу.

Примітка: вам потрібно досить багато пам'яті для виконання цієї операції, оскільки тимчасовий індексний файл створюється в пам'яті і тільки потім переписується на диск.

1.23 Locate

function Locate(const KeyFields: string; const KeyValues: Variant;

Options: TLocateOptions): Boolean; override;

Функція Locate здійснює пошук в наборі даних запису і позиціонує вказівник не неї. Параметр KeyFields це рядок, що містить список імен полів, розділених символом крапка з комою, за якими проводиться пошук. Параметр KeyValues це варіантний масив, що містить значення пошуку для полів. Якщо список KeyFields одиночне поле, параметр KeyValues вказує значення для цього поля для необхідної запису. Для вказівки множинних значень пошуку, передайте варіантний масив як KeyValues.

Параметр Options це безліч, яке вказує параметри пошуку по строкових полях. Якщо параметр Options містить loCaseInsensitive, функція Locate ігнорує регістр букв при порівнянні полів. Якщо Options містить loPartialKey, функція Locate дозволяє частковий збіг з рядком в KeyValues. Якщо Options порожня множина, або якщо властивість KeyFields не містить строкових полів, то параметр Options ігнорується. Функція Locate повертає true якщо шукана запис знайдено, і робить цей запис поточної, інакше функція повертає false. Якщо вказано одне поле і вона міститься в індексному файлі, то пошук використовує даний індекс. У цьому випадку Options ігнорується, як якби Ви вказали loPartialKey.

1.24 LocateRecord

function LocateRecord(const KeyFields: string; const KeyValues: Variant; Options: TLocateOptions; bSyncCursor: Boolean): Boolean;

Це внутрішній метод, який виробляє всю дійсну роботу з пошуку.

1.25 IsDeleted

function IsDeleted: Boolean;

Викличте функцію IsDeleted для перевірки, що поточна запис позначена як віддалена. Ця функція діє, якщо властивість ShowDeleted встановлено у true.

1.26 Undelete

procedure Undelete;

Викличте процедуру Undelete для зняття позначки про видалення.

1.27 CreateTable

procedure CreateTable;

Викличте процедуру CreateTable під час виконання для створення таблиці, використовуючи поточні визначення набору даних. Якщо таблиця вже існує, то процедура CreateTable перезапише структуру таблиці і дані. Якщо властивість FieldDefs містить значення, ці значення використовуються для створення визначень полів. Інакше буде використано властивість Fields. Одне або декілька з цих властивостей може містити значення для створення таблиці. Якщо властивість Indexes містить значення, то ці значення використовуються для створення індексів у таблиці. Див Також CreateTableEx.

1.28 CreateTableEx

procedure CreateTableEx(DbfFieldDefs: TDbfFieldDefs);

Викличте процедуру CreateTableEx для створення таблиці, використовуючи дані визначення полів. Дані визначення полів дають більше можливостей, наприклад, таких як визначення точності цифрових полів.

1.29 CopyFrom

procedure CopyFrom(DataSet: TDataSet; FileName: string;

DateTimeAsString: Boolean; Level: Integer);

Використовуйте дану процедуру для копіювання заданого набору даних в нову таблицю. Параметр DataSet це тип TDataSet, з якого ви хочете скопіювати, FileName це повне ім'я нової таблиці (включаючи шлях і розширення). DateTimeAsString вказує, що поля datetime повинні бути перетворені в рядкові поля в таблиці результату. Це особливо корисно якщо ви бажаєте використовувати TDbf для створення до текстових процесорів наприклад. Якщо цей параметр встановлений в True, то спрацьовує обробник події OnCopyDateTimeAsString, де ви можете перекрити стандартне перетворення з datetime в string, яке базується на поточних регіональні настройки. Властивість Level визначає TableLevel для нової таблиці.

У разі використання версій TDbf до 6.0 для перетворення datetime значень в BDE сумісний формат, використовуйте цю процедуру наступним чином: помістіть два примірники TDbf на форму, встановіть у TDbf1 властивість DateTimeHandling в dtDateTime і підключіть до існуючої таблиці. Переконайтеся, що у TDbf2 воно встановлено в dtBDETimeStamp і викличте процедуру CopyFrom з DataSet = TDbf1 і DateTimeAsString = False. Потім Ви можете замінити стару таблицю новою і використовувати TDbf в режимі dtBDETimeStamp у ваших додатках.

1.30 RestructureTable

procedure RestructureTable(DbfFieldDefs: TDbfFieldDefs; Pack: Boolean);

Викличте процедуру RestructureTable для зміни структури поточної таблиці.

Параметр DbfFieldDefs дозволяє вказати нову структуру. Кожен елемент FieldDef містить властивість CopyFrom, яке є індексом поля з якого копіюється інформація. Поля, з які не вказується в жодному з DbfFieldDef властивостей CopyFrom, видаляються. Якщо ви призначили копію визначення таблиці DbfFieldDefs новому списку для модифікації, то властивість CopyFrom нового списку FieldDefs автоматично призначається, виключаючи Delphi 3 користувачів. Тому користувачі Delphi 3 повинні потурбуватися призначенням властивості CopyFrom для запобігання видалення полів, чого природно ви не хочете! Параметр Pack вказує необхідність упаковки таблиці, для видалення записів позначених для видалення.

1.31 PackTable

procedure PackTable;

Викличте процедуру PackTable для остаточного видалення записів позначених для видалення. Перед тим як ці записи будуть остаточно видалені, вони позначаються до видалення. Виклик процедури PackTable видаляє ці записи фізично з таблиці. Також це можна зробити за допомогою виклику процедури RestructureTable з покажчиком nil для установки DbfFieldDefs і передачею true для Pack.

1.32 EmptyTable

procedure EmptyTable;

Метод EmptyTable видаляє всі записи з таблиці. Він не змінює поточну структуру таблиці та індексів.

1.33 Zap

procedure Zap;

Псевдонім для методу EmptyTable.

1.34 InitFieldDefsFromFields

{$ifndef DELPHI\_5}

 procedure InitFieldDefsFromFields;

{$endif}

Це внутрішній метод InitFieldDefsFromFields, що використовується різними функціями, наприклад, CreateTable. Він створює визначення полів для існуючого набору Fields. Дана процедура нудна тільки для Delphi 4 і молодше, оскільки в Delphi 5 і старше дана процедура реалізована в TDataSet.

2. Властивості

2.1 AbsolutePath

property AbsolutePath: string read FAbsolutePath;

Це абсолютний шлях до поточної таблиці. Див також FilePathFull.

2.2 DbfFieldDefs

property DbfFieldDefs: TDbfFieldDefs read GetDbfFieldDefs;

DbfFieldDefs це список визначень полів для таблиці, подібний до TDataSet.FieldDefs, за винятком того, що TDbfFieldDefs більш підходить для dBase таблиць. Наприклад, він включає інформацію про рідних типах полів і точність для числових полів. Див також TDbfFieldDefs. Для доступу до полів і їх значень в наборі даних використовуйте властивості Fields і Field Values, а також метод FieldByName.

2.3 PhysicalRecNo

property PhysicalRecNo: Integer read GetPhysicalRecNo write SetPhysicalRecNo;

Використовуйте властивість PhysicalRecNo для визначення фізичного номера запису поточного запису. Його також можна використовувати для встановлення курсору на потрібний запис. Різниця між RecNo полягає в тому, що читання RecNo повертає послідовний номер, який той же самий якщо індекс не використовується і розрізняється якщо індекс використовується.

2.4 LanguageID

property LanguageID: Integer read GetLanguageID;

Використовуйте властивість LanguageID для визначення кодової сторінки, комбінації про використану локалізації using. Див також DbfLang.pas для отримання інформації.

2.5 LanguageStr

property LanguageStr: String read GetLanguageStr;

Використовуйте властивість LanguageStr для читання кодової сторінки, інформації про локалізацію для dBase таблиць рівня 7.

2.6 CodePage

property CodePage: Cardinal read GetCodePage;

Використовуйте властивість CodePage для визначення кодової сторінки dBase таблиці.

2.7 ExactRecordCount

property ExactRecordCount: Integer read GetExactRecordCount;

Використовуйте властивість ExactRecordCount для визначення точної кількості записів у поточному наборі даних. Воно включає в себе кількість видалених, відфільтрованих і індексованих записів. Цим воно відрізняється від властивості RecordCount, яке завжди дає існуючу верхньої межі. Зауважимо, що дана властивість вимагає сканування повного набору даних, у той час як властивість RecordCount це просто розрахунок.

2.8 DbfFile

property DbfFile: TDbfFile read FDbfFile;

Ця властивість використовується внутрішньо для функцій нижнього рівня.Додатку користувача немає потреби звертатися до даного властивості.

2.9 DisableResyncOnPost

property DisableResyncOnPost: Boolean read FdisableResyncOnPost

 write FDisableResyncOnPost ;

Коли запис записується в таблицю, TDataSet витягує всі записи поблизу від поточного запису. Властивість DisableResyncOnPost керує даним поведінкою. Воно дозволяє збільшити швидкість, якщо Ви додаєте групу записів. Див також TDataSet.DisableControls.

2.10 DateTimeHandling

property DateTimeHandling: TDateTimeHandling read FdateTimeHandling

 write FDateTimeHandling default dtBDETimeStamp;

До версії 6.0 TDbf використовував запис значення в '@' (ftDateTime) полів як Delphi тип TDateTime. Для сумісності з BDE, ці поля треба було записувати як BDE тип TimeStamp (число мілісекунд, починаючи з 01/01/0001 плюс один день). Для забезпечення сумісності Ви можете використовувати цю властивість для визначення як TDbf повинен читати і писати значення datetime як TDateTime або як BDE TimeStamp. Зараз за замовчуванням це як dtBDETimeS – tamp, але для читання значень в існуючих TDbf таблицях Ви можете вибрати тип dtDateTime. Якщо ви бажаєте перетворити ваші дані, сумісні з BDE, то зверніть вашу увагу на процедуру CopyFrom.

2.11 Exclusive

property Exclusive: Boolean read FExclusive write FExclusive default false;

Використовуйте властивість Exclusive для запобігання доступу інших додатків до таблиці, відкритої в цьому додатку. Перед відкриттям таблиці, встановіть властивість Exclusive в true. Таблиця повинна бути закрита до зміни властивості Exclusive. Коли властивість Exclusive встановлено у true, і додаток успішно відкрило таблицю, то інші програми не зміг мати до неї доступу. Якщо таблиця, до якої потрібно монопольний доступ, вже відкрита іншим додатком, то виникне виняток. Для обробки таких винятків, укладіть відкриття в блок try except end. Див також TryExclusive. Не встановлюйте властивість Exclusive в true під час розробки, якщо Ви одночасно встановлюєте властивість Active в true. В цьому випадку відбудеться виключення, оскільки таблиця вже використовується в IDE.

2.12 FilePath

property FilePath: string read FRelativePath write SetFilePath;

Використовуйте властивість FilePath для визначення, який шлях встановив користувач до файлу поточної таблиці. Він може бути або відносним або абсолютним. Див також FilePathFull.

2.13 FilePathFull

property FilePathFull: string read FAbsolutePath write SetFilePath

stored false;

Використовуйте властивість FilePathFull для визначення абсолютного шляху для поточної таблиці. Воно завжди містить абсолютний шлях, незалежно який шлях вказаний у властивості FilePath. Звичайно це використовується під час розробки в IDE, коли ви встановлюєте відносний шлях у властивості FilePath, потім перевіряєте властивість FilePathFull щоб визначити де буде створена чи відкрита таблиця.

2.14 Indexes

property Indexes: TDbfIndexDefs read FIndexDefs write SetDbfIndexes;

Властивість Indexes це колекція визначень індексів, кожен елемент якої описує окремий індекс для таблиці. Визначайте визначення індексу до виклику процедури CreateTable або створення таблиці під час розробки в середовищі. Зазвичай додаток робить доступ або вказує індекси під час виконання, через властивості IndexName і IndexFieldNames. Якщо властивість Indexes оновлюється або вручну редагується, то властивість StoreDefs встановлюється в true. Визначення індексу у властивості Indexes не завжди відображає поточні індекси доступні для таблиці. Перед перевіркою властивості Indexes, викличте метод Update для оновлення списку.

2.15 IndexFieldNames

property IndexFieldNames: string read GetIndexFieldNames

write SetIndexFieldNames;

Використовуйте властивість IndexFieldNames як альтернативний метод для визначення індексу використовуваного для таблиці. У властивості IndexFieldNames, вкажіть ім'я кожної використовуваної колонки як індекс для таблиці. Ви можете також вказати вираз для існуючого індексу. Назва колонки, зазначене у властивості IndexFieldNames повинно бути вже проіндексовано. Властивості IndexFieldNames і IndexName взаємно виняткові. Установка одного очищає інше.

2.16 IndexName

property IndexName: string read GetIndexName write SetIndexName;

Використовуйте властивість IndexName для вказівки альтернативного індексу для таблиці. Якщо властивість IndexName пусто, то сортування здійснюється з фізичного порядку записів у таблиці. Якщо IndexName містить ім'я допустимого індексу, то індекс визначає сортування записів. Назва індексу, зазначене у властивості IndexName має бути або в головному індексному файлі таблиці або в іншому індексному фото, вже зазначеному у властивості Indexes або відкритому за допомогою функції OpenIndexFile. IndexFieldNames і IndexName є взаємовиключними. Налаштування одного очищає інше.

2.17 MasterFields

property MasterFields: string read GetMasterFields write SetMasterFields;

Використовуйте властивість MasterFields після установки властивості MasterSource вкажіть назву одне або більше полів для створення зв'язку основна підпорядкована між цією таблицею та таблицею зазначеної у властивості MasterSource. Властивість MasterFields це рядок, що містить одну або більше імен полів в основній таблиці. Поля розділяються символів крапку з комою. Кожен раз, коли поточний запис змінюється в головній таблиці, то нове значення цих полів використовується для вибору відповідних записів у підпорядкованій таблиці.

2.18 MasterSource

property MasterSource: TDataSource read GetDataSource write SetDataSource;

Використовуйте властивість MasterSource вкажіть назву компонента, чиє властивість DataSet використовується як головна таблиця для встановлення зв'язку з підлеглою.Вказане властивість DataSource має бути інший TDbf таблицею. Під час розробки виберіть доступне джерело з меню, що випадає властивості MasterSource в інспектора об'єктів. Після установки властивості MasterSource вкажіть, які поля використовуються в головній таблиці через установку властивості MasterFields. Під час виконання, кожен раз, коли поточний запис у головній таблиці змінюється, нове значення використовується для вибору відповідних записів у підпорядкованій таблиці.

2.19 OpenMode

property OpenMode: TDbfOpenMode read FOpenMode write FopenMode

default omNormal;

Властивість OpenMode вказує, що робити, якщо зазначена з даним ім'ям таблиці не існує і властивість Active в встановлюється істинне або викликається метод відкритим.

* Значення omNormal перериває відкриття, якщо файл не існує.
* Значення omAutoCreate створює нову таблицю, якщо викликається метод CreateTable і відкриває її.
* Значення omTemporary не використовується.

2.20 ReadOnly

property ReadOnly: Boolean read FReadOnly write FReadonly

default false;

Властивість ReadOnly вказує, що таблицю треба відкрити в режимі тільки читання. Якщо це властивість встановлено в True, то таблицю не можна змінювати. Ви можете відкрити таблицю в режимі тільки читання, навіть якщо вона відкрита в монопольному режимі.

2.21 ShowDeleted

property ShowDeleted: Boolean read FShowDeleted write SetShowDeleted

 default false;

Властивість ShowDeleted вказує показувати записи, позначені для видалення чи ні. Використовуйте функцію IsDeleted для визначення, що поточна запис позначена для видалення.

2.22 Storage

property Storage: TDbfStorage read FStorage write FStorage default stoFile;

Властивість не використовується.

2.23 StoreDefs

property StoreDefs: Boolean read FStoreDefs write FStoreDefs default False;

StoreDefs якщо властивість встановлено в True, то визначення індексів таблиці та її полів записуються разом з модулем даних або формою. Установка властивості StoreDefs в істинно робить можливим створення полів, індексів методом CreateTable простою справою, за один дзвінок під час виконання програми. Властивість StoreDefs за замовчуванням встановлено в помилковими. Воно встановлюється в Щоправда, коли властивості FieldDefs або індекси оновлюється або редагується вручну; для запобігання запису змінених або імпортованих визначень не забудьте скинути це властивість в помилковими.

2.24 TableName

property TableName: string read FTableName write SetTableName;

Використовуйте властивість TableName вкажіть назву файлу таблиці бази даних. Ви можете вказати або повний шлях з ім'ям файлу, в цьому випадку шлях буде відділений і записаний в властивість FilePath. При установці TableName, властивість Active повинно бути в змозі помилковою.

2.25 TableLevel

property TableLevel: Integer read FTableLevel write SetTableLevel;

Використовуйте властивість TableLevel для визначення рівня поточної таблиці. Установка властивості TableLevel дозволяє вказати рівень для знову створюваних таблиць.Властивість Active повинно бути в змозі помилкової під час встановлення цієї властивості.Існують такі рівні:

* сумісність з DBase III +
* сумісність з DBase IV. Єдина різниця в DBase III + це поточна кодова сторінка, локалізація DBase III + не використовує перетворення для кодової сторінки і використовує двійковий порядок для сортування.
* сумісність з Visual DBase VII. Не всі властивості підтримані, але тут наведено підтримані:
	+ Більше типів полів: DateTime, 32-бітні цілі, 64-бітові подвійної точності.
	+ Значення за замовчуванням для полів. Ця інформація доступна через властивості DbfFieldDef, .HasDefault і DefaultBuf.
	+ Значення min і max для полів не підтримані, але можуть бути прочитані.
	+ Посилальна цілісність не підтримана.
* Сумісність з FoxPro. Ці типи трохи розрізняються, але дуже сумісні з DBase IV. CDX індекси не підтримані.

2.26 UseFloatFields

property UseFloatFields: Boolean read FUseFloatFields write FuseFloatFields

default true;

Коли властивість UseFloatFields дозволено, то це змушує використовувати поля з плаваючою комою, навіть якщо числові поля мають нульову точність. Коли заборонено, то використовуються 32 або 64 бітові цілочисельні поля, в залежності від розміру поля.

2.27 Version

property Version: string read GetVersion write SetVersion stored false

Використовуйте властивість версії для визначення версії TDbf компонент.

2.28 BeforeAutoCreate

property BeforeAutoCreate: TBeforeAutoCreateEvent read FbeforeAutoCreate

 write FBeforeAutoCreate;

Коли таблиця відсутня, а властивість встановлено в OpenMode omAutoCreate і викликаний метод відкритим, то порушується ця подія. Реалізуйте обробник BeforeAutoCreate для запобігання створення таблиці.

2.29 OnCompareRecord

property OnCompareRecord: TNotifyEvent read FOnCompareRecord

 write FOnCompareRecord;

Дана подія не використовується.

2.30 OnLanguageWarning

property OnLanguageWarning: TLanguageWarningEvent read FOnLanguageWarning

 write FOnLanguageWarning;

Напишіть обробник OnLanguageWarning для заборони дій, коли дані записуються в таблицю зі специфічною кодовою таблицею, а операційна система не в змозі перетворити дані для перегляду в кодову сторінку ANSI. Ви можете вказати режим тільки читанні або редагувати ні на що не дивлячись.

2.31 OnLocaleError

property OnLocaleError: TDbfLocaleErrorEvent read FOnLocaleError

 write FOnLocaleError;

Напишіть обробник OnLocaleError для заборони дій, коли індексні дані записані в певному порядку, а операційна система не підтримує цей порядок сортування. Ви можете читати або змінювати індекс ні на що не дивлячись, але індекс може виявитися пошкодженим якщо Ви не розумієте, що робите.

2.32 OnIndexMissing

property OnIndexMissing: TDbfIndexMissingEvent read FOnIndexMissing

 write FOnIndexMissing;

Напишіть обробник OnIndexMissing для заборони дій, коли таблиця посилається на конкретний індекс, але індекс відсутній. Значення за замовчуванням це видалити зв'язок.

2.33 OnCopyDateTimeAsString

property OnCopyDateTimeAsString: TConvertFieldEvent

 read FOnCopyDateTimeAsString write FOnCopyDateTimeAsString;

Напишіть обробник OnCopyDateTimeAsString для надання свого форматування полів DateTime в рядкові поля. Див також процедуру CopyFrom.

2.34 OnTranslate

property OnTranslate: TTranslateEvent read FOnTranslate write FOnTranslate;

Напишіть обробник OnTranslate для надання користувача перетворення даних таблиці в кодову таблицю "ANSI".